MFC 的消息处理机制

Windows 应用程序依靠外部发生的事件来驱动，程序一直等待任何可能的输入，做出判断再进行处理。所有的 GUI 系统都是以消息为基础、以事件为驱动。

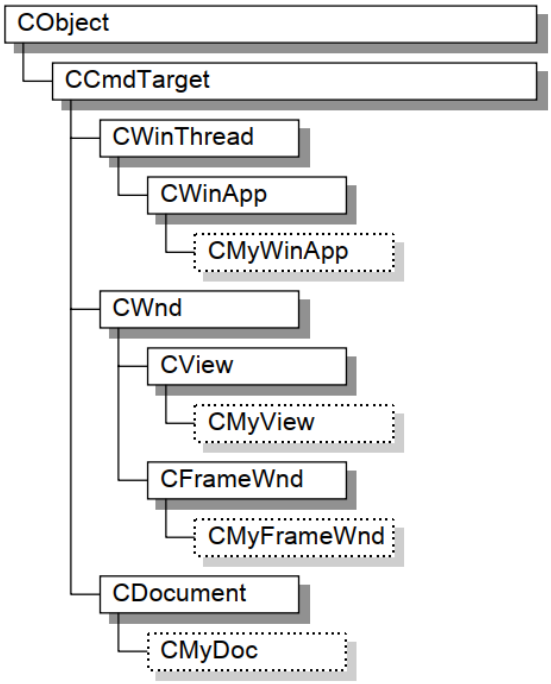
每一个 windows 应用程序都应该有一个如下回路：

```
while (GetMessage(&msg, ……)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

其中，msg 是一种 windows 内定的一种格式：

```
typedef struct tagMSG{
    HWND hwnd;
    UINT message; // WM_xxx，例如 WM_MOUSEMOVE，  WM_SIZE...
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
} MSG;
```

由窗口接受并处理消息。每一个窗口都有一个负责处理消息的函数，程序员必须负责设计这个所谓的「窗口函数」（ window procedure，或称为 window function）。如果窗口获得一个消息，这个窗口函数必须判断消息的类别，决定处理的方式（消息映射）。



依靠宏实现的消息映射：

在类声明中：

```
class CScribbleDoc : public CDocument
{
    ...
    DECLARE_MESSAGE_MAP()
};
```

在类源码中：

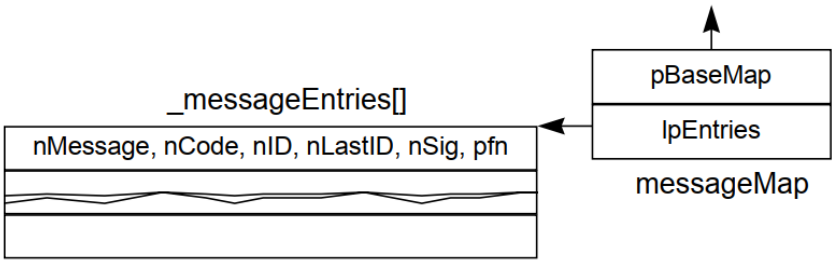
```
BEGIN_MESSAGE_MAP(CScribbleDoc, CDocument)
//{{AFX_MSG_MAP(CScribbleDoc)
ON_COMMAND(ID_EDIT_CLEAR_ALL, OnEditClearAll)
ON_COMMAND(ID_PEN_THICK_OR_THIN, OnPenThickOrThin)
...
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
#define DECLARE_MESSAGE_MAP() \
private: \
    static const AFX_MSGMAP_ENTRY _messageEntries[ ]; \
protected: \
    static AFX_DATA const AFX_MSGMAP messageMap; \
    virtual const AFX_MSGMAP* GetMessageMap() const; \


struct AFX_MSGMAP_ENTRY
{
  UINT nMessage;        // windows message
  UINT nCode;
  UINT nID;
  UINT nLastID;
  UINT nSig;
  AFX_PMSG pfn;   // routine to call (or special value)
};
```

其中，**AFX_PMSG** 是函数指针类型

```
struct AFX_MSGMAP
{
  const AFX_MSGMAP* pBaseMap;
  const AFX_MSGMAP_ENTRY* lpEntries;
};
```



```
#define BEGIN_MESSAGE_MAP(theClass, baseClass) \
const AFX_MSGMAP* theClass::GetMessageMap() const \
{ return &theClass::messageMap; } \
AFX_DATADEF const AFX_MSGMAP theClass::messageMap = \
{ &baseClass::messageMap, &theClass::_messageEntries[0] }; \
const AFX_MSGMAP_ENTRY theClass::_messageEntries[] = \
{ \
#define END_MESSAGE_MAP() \
{0, 0, 0, 0, AfxSig_end, (AFX_PMSG)0 } \
}; \
```

AfxSig_end 在 AFXMSG_.H 中被定义为 0

```
#define ON_COMMAND(id, memberFxn) \
{ WM_COMMAND, CN_COMMAND, (WORD)id, (WORD)id, AfxSig_vv,
(AFX_PMSG)memberFxn },
#define ON_WM_CREATE() \
{ WM_CREATE, 0, 0, 0, AfxSig_is, \
(AFX_PMSG)(AFX_PMSGW)(int (AFX_MSG_CALL
CWnd::*)(LPCREATESTRUCT))OnCreate },
#define ON_WM_DESTROY() \
{ WM_DESTROY, 0, 0, 0, AfxSig_vv, \
(AFX_PMSG)(AFX_PMSGW)(void (AFX_MSG_CALL CWnd::*)(void))OnDestroy },
#define ON_WM_MOVE() \
{ WM_MOVE, 0, 0, 0, AfxSig_vvii, \
(AFX_PMSG)(AFX_PMSGW)(void (AFX_MSG_CALL CWnd::*)(int, int))OnMove },
```

消息流动

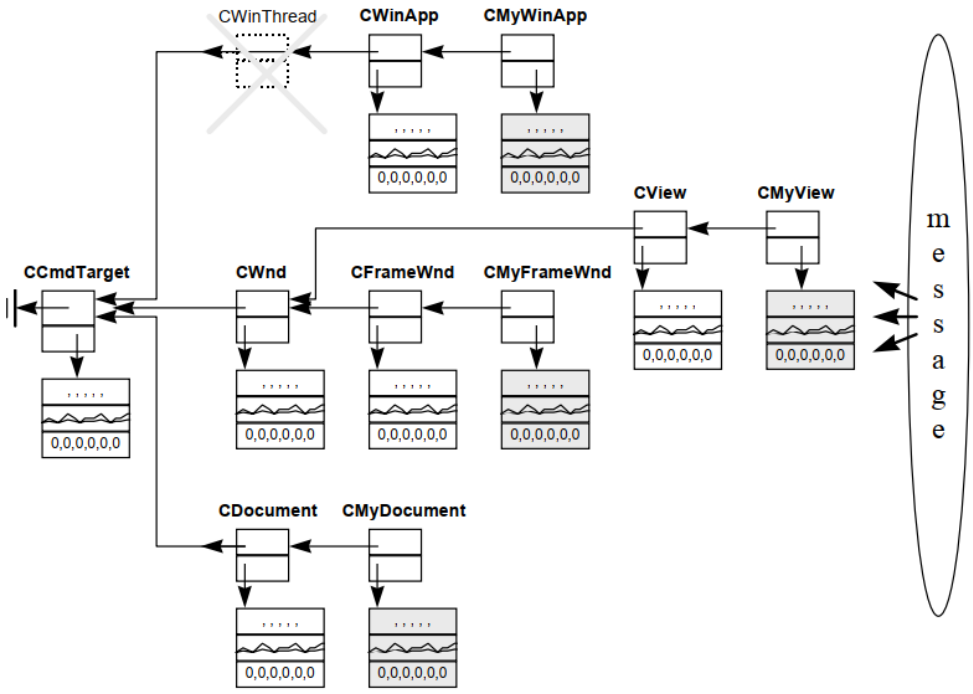


图9-2 MFC 消息映射表（ 也就是消息流动网）

对于命令消息 WM_COMMAND 的特殊处理顺序：

命令消息接收物的类型	处理次序
Frame 窗口	1. View 2. Frame 窗口本身 3. CWinApp 对象
View	1. View 本身 2. Document
Document	1. Document 本身 2. Document Template

图9-4 MFC 对于命令消息WM_COMMAND 的特殊处理顺序。